
FreeBSD™ JOURNAL
Jan/Feb 2018
STORAGE
A Review of Storage Multipathing
Tape's Not Dead
(And May Even Be a Good Solution)
AND ALSO...
TRACING ifconfig Commands
from User Space to Device Driver

BORN TO DISRUPT

MODERN. UNIFIED. ENTERPRISE-READY.

INTRODUCING THE TRUENAS® X10, THE MOST COST-EFFECTIVE ENTERPRISE STORAGE ARRAY ON THE MARKET.

Perfectly suited for core-edge configurations and enterprise workloads such as backups, replication, and file sharing.

- ★ **Modern:** Not based on 5-10 year old technology (yes that means you legacy storage vendors)
- ★ **Unified:** Simultaneous SAN/NAS protocols that support multiple block and file workloads
- ★ **Dense:** Up to 120 TB in 2U and 360 TB in 6U
- ★ **Safe:** High Availability option ensures business continuity and avoids downtime
- ★ **Reliable:** Uses OpenZFS to keep data safe
- ★ **Trusted:** Based on FreeNAS, the world's #1 Open Source SDS
- ★ **Enterprise:** 20TB of enterprise-class storage including unlimited instant snapshots and advanced storage optimization for under $10,000

The new TrueNAS X10 marks the birth of a new entry class of enterprise storage. Get the full details at iXsystems.com/TrueNAS.

iXsystems™

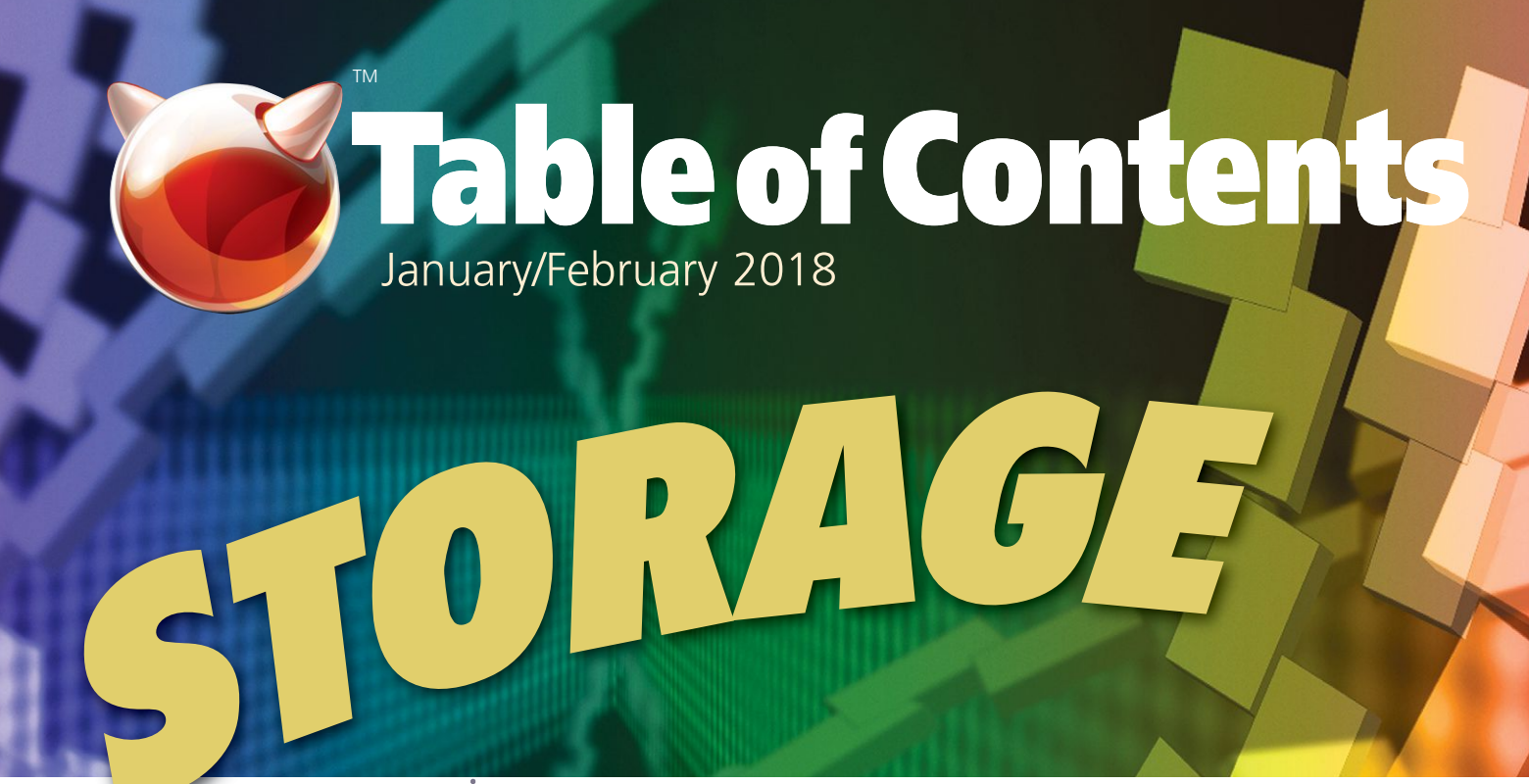


Table of Contents

January/February 2018

STORAGE

ALSO INSIDE...

Support FreeBSD®

Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.
freebsdfoundation.org/donate

FreeBSD™
FOUNDATION

S&W Publishing LLC
PO BOX 408, BELFAST, MAINE 04915

Publisher • Walter Andrzejewski
walter@freebsdjournal.com

Editor-at-Large • James Maurer
jmaurer@freebsdjournal.com

Copy Editor • Annaliese Jakimides

Art Director • Dianne M. Kischitz
dianne@freebsdjournal.com

Office Administrator • Michael Davis
davism@freebsdjournal.com

Advertising Sales • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).
Published by the FreeBSD Foundation,
5757 Central Ave., Suite 201, Boulder, CO 80301
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsdfoundation.org

LETTER from the Board

With this January/February issue we get to celebrate the New Year twice—once on January 1, and again on February 16, the Lunar New Year celebrated throughout much of Asia—including Taiwan, where the first BSDTW conference was held this past November. I recommend Brooks Davis's excellent report in this issue, and I'd also like to thank the organizers of BSDTW for the amazing job they did in developing that conference from concept to the opening talk in less than a year.

Organizing a conference requires a staggering amount of work. Pick dates that don't overlap with too many competing events, get a venue at a reasonable price, figure out how to provide food for 100 to 200 hungry engineers, and build a program committee—all are just parts of the process. And, oh yes, there is the funding. Open-source code may be free as in beer, but let me tell you the beer you drink at a conference is not free. More than half the work of conference organizing is finding companies and attendees to pay for it.

BSDTW was the brainchild of some FreeBSD developers in Taiwan, Li-Wen Hsu, Ruey-Cherng Yu, and Ray Chen, who approached me over tea in December 2016 after a talk I'd given at National Taiwan University. The tea was just an excuse to sit down and try to convince me and, by proxy, the FreeBSD Foundation to support a conference in Taiwan. It didn't take much convincing since I knew they'd be doing a lot of the work and all I had to do was persuade the Foundation Board to put in some seed money. As it happens, I wound up running the Program Committee, but it was the folks in Taiwan who turned the idea into butts in seats by November 2017. BSDTW was perhaps the most successful first-time conference ever held by the BSDs. When it was time for that first keynote talk, all the seats in the venue had been sold out, and they were selling standing room tickets. The conference went off without a hitch—it was an amazing performance—and I hope they'll do it again in 2018.

Now, let's turn to our slate of articles in this issue, which includes two excellent pieces about storage. Alexander Motin's article describes how multipathing works to improve the reliability and performance of the storage subsystems in FreeBSD, and Ken Merry writes about tape systems, which, he asserts, are not dead. Farhan Kahn walks us through just what happens when we use the ifconfig(8) command, which is what controls all network devices on FreeBSD. Tracing ifconfig(8) allows Farhan to show us many interacting pieces of the networking code, and since his main focus is wireless devices, there is quite a maze of twisty little passages to follow.

Columns include Brooks Davis's previously mentioned BSDTW report, a storage-related book review by Matt Joras, an svn Update installment by Steven Kreuzer, and Events Calendar from Dru Lavigne. Speaking of events, the next BSD-related conference is AsiaBSD, March 8–11, 2018, in Tokyo, Japan, and I look forward to seeing everyone there!

George Neville-Neil
President of the *FreeBSD Foundation* Board of Directors

SEE TEXT ONLY

A REVIEW OF STORAGE Multipathing

BY ALEXANDER MOTIN

Storage multipathing is a technique designed to improve storage reliability by eliminating a single point of failure, and/or improving its performance by distributing workload between multiple physical connections. Multipathing can be implemented on several different layers: the transport layer (like iSCSI), the peripheral device driver layer (like the SCSI block device driver), or the block storage transformation layer (like FreeBSD GEOM). Each implementation has its own benefits and downsides.

Transport layer multipathing allows multiple physical connections to be exposed as one transport protocol connection. For example, the iSCSI protocol defines optional support for multiple connections per session (MC/S), when SCSI requests and responses are distributed between several TCP connections and are visible to upper SCSI layers as a single initiator to the target connection. This technique can be useful if initiator and/or target storage stacks do not natively support application layer multipathing. For example, Microsoft supports SCSI layer multipathing (MPIO) only in the server version of Windows, while iSCSI MC/S is fully supported even for desktops. The biggest downside of this technique is that only paths using the same capable transport protocol (like iSCSI) may participate—you cannot back up Fibre Channel fabric with iSCSI this way. Also, the requirement to provide semantics of a single SCSI initiator-target connection (request ordering, error recovery, etc.) significantly complicates the transport protocol implementation. For example, if one of the connections experiences delays or packet losses, other connections have to delay their otherwise already delivered requests while that connection recovers, or they must implement mechanisms for quick problem detection with the affected requests being rerouted via different connections. For these reasons, the new FreeBSD kernel space, the iSCSI initiator and target implementations do not support MC/S. The old user space iSCSI target implementation, istgt, declared MC/S support, but implemented it in a minimalistic way without proper error recovery, and that sometimes improved storage performance, but could also cause suffering on the reliability side.

Peripheral device driver-layer multipathing allows the execution of upper-layer requests (such as block read, write, delete, etc.) via multiple, transport-layer connections (such as iSCSI, Fibre Channel, SAS, etc.). This technique permits simultaneous use of different transport protocols only limiting to a single application protocol (such as SCSI Block Commands). You cannot backup an NVMe-over-Fabric connection with iSCSI using multipathing at this layer unless you translate one command set (NVMe) into another (SCSI). Multipathing at this layer assumes a completely different view of request execu-

tion ordering. Since the target does not know which of the multiple initiators represent the same peripheral device driver (as the source of properly ordered requests), it can not guarantee ordering for requests coming via different paths. This significantly simplifies multipathing-capable target implementation, but requires a special compilation of request distribution logic between the paths in the peripheral driver, which should prevent, or at least minimize, unwanted request reordering, probably at the cost of lower path utilization. Fortunately, the peripheral driver knows more about the nature and origination of requests, and can do it in a more clever way than is possible on transport layers.

For people familiar with network protocols, an analogy can be drawn: transport-layer multipathing is akin to Multilink PPP, which guarantees full, original ordering of all packets as required by upper layers of the PPP stack, and distributes packet fragments between the available links without looking inside; same-time, peripheral device driver-layer multipathing can be compared to LACP, which identifies individual data streams, looking at a packet's MAC and IP addresses, TCP/UDP ports, etc., to distribute them between the links in a way that would guarantee ordering, only separately within each stream. While the first approach may theoretically provide much better throughput for a low number of data streams, the second provides a less complicated stateless implementation, lower average latency, and better resiliency to individual link problems.

As an example of minimal, peripheral driver-layer multipathing implementation, we may look at SAS HDDs. Each SAS HDD typically has two SAS ports, each of which can be connected to a separate SAS expander of the backplane, and then to a separate HBA of the host, providing the host two separate SCSI `I_T_L` (Initiator-Target-Logical Unit) nexuses. A multipathing-capable SCSI peripheral driver can identify those two `I_T_L` nexuses as paths to the same Logical Unit by fetching and comparing the globally unique Logical Unit Name (can be NAA or EUI IDs, text or binary strings, UUIDs, etc.). After that, the peripheral driver can report a single block storage device to upper layers and choose a strategy to distribute workload between those two paths. Since, for a typical SAS HDD interface, throughput is much higher than media throughput, and sequential access is much faster than random, simple failover configuration, using only one of the paths at a time can be perfect. The same time for high-performance SSDs throughput of a single SAS link may not be enough to saturate the backing storage, requiring both paths to be utilized simultaneously to reach full performance.

If peripheral device-driver multipathing is not supported or cannot be used, this functionality can be implemented on a higher, block-storage transformation layer, such as GEOM on FreeBSD or Device Mapper on Linux. This layer operates in terms of abstract block read, write, delete, etc., requests, and is not related to the specifics of any transport or application protocol. It may use some out-of-band information from the lower layers or work independently. Better integration makes it possible to automate its operation, and, thus, makes it more reliable, but it also makes the storage stack more complicated. The FreeBSD GEOM MULTIPATH class, by default, operates without using any out-of-band information, relying only on its own metadata, stored at the last sector of the device, and the configuration provided by administrator. But it also has a mode of operation without using the on-disk metadata, relying on an external government of some third-party code to manage path detection, activation, prioritization, failing, etc., and by relying on the Logical Unit Names and other information.

Things become much more interesting when target ports are not equal. SCSI specifications call it Asymmetric Logical Unit Access (ALUA). ALUA can be caused by different things, but typically it is caused by the presence of several storage controllers within the target device that handle requests from different target ports, and which, for some reason, cannot work simultaneously either at all, or for some Logical Units. The degree of the asymmetricity may vary. In the worst case, some ports/controllers may be unable to even report any identification information for the Logical Unit. ALUA calls this state «`Unavailable`». This is not a very useful state, since it does not provide sufficient information even for automatically setting up multipathing. The next, better state is called «`Standby`». In this state, the port is able to report full identification of the Logical Unit, handle SCSI reservation and some management requests, but is unable to access the media. This state allows automatic multipathing configuration, but also demands full ALUA multipathing support from the initiator OS, since data access requests sent via the wrong path will fail. The next state is called «`Active/Non-optimized`». This state allows full Logical Unit access via the

path, but the performance characteristics of such access may be more or less suboptimal, for example, due to the need for additional synchronization or command/data transfer between the storage controllers. This state allows the Logical Unit to be used, even with a peripheral driver without ALUA support, but such use may be inefficient if the wrong path is selected. And finally, the best state the port may have is «`Active/Optimized`». This state means that the Logical Unit is fully accessible via this port with maximal performance. An additional ALUA state is called «`Transitioning`» and covers situations when the port or the Logical Unit is changing its state and is temporarily not accessible. The duty of the ALUA-capable initiator is to continuously monitor the state of each Logical Unit and choose the optimal path through which to send requests. In some cases, the ALUA-capable initiator may explicitly or inexplicitly request port state transitions, but the logic of those requests is outside the scope of SCSI specifications and is vendor-specific.

The multipathing operation with ALUA can be exemplified with the TrueNAS storage appliance from iXsystems Inc., which uses the High-Availability functionality of the FreeBSD CTL subsystem to implement the ALUA-capable multipath SCSI target. The TrueNAS High-Availability appliance includes two storage controllers, each running its own copy of FreeBSD 11, connected via either Ethernet or by a PCIe Non-Transparent Bridge internal link, and having access to a shared array of SAS disks. The appliance uses ZFS pools to store the user data, but since ZFS is not a clustered filesystem, only one of the controllers can access a specific pool at any point in time. SCSI requests received via iSCSI or the Fibre Channel links with the controller having no access to the required pool are proxied by CTL to the other controller via the internal link. Providing multipathing functionality without ALUA in such a situation would cause either additional latency or severe bandwidth limitations, depending on the type of internal link used. The use of ALUA allows a capable multipathing client to know the present situation and always submit requests via the most efficient paths. Let's look at some practical examples:

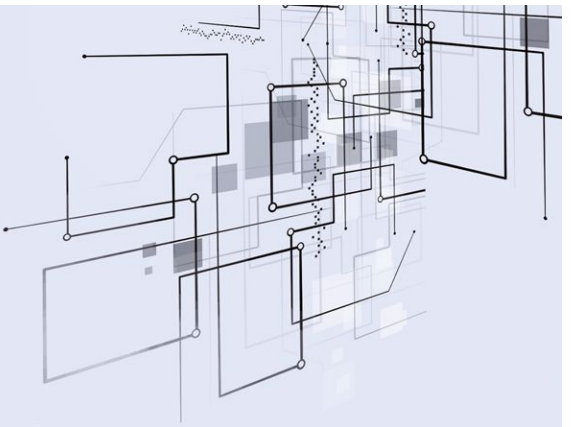
A storage system with two controllers was configured to provide one iSCSI target Logical Unit in ALUA mode. Multipath-related parts of the CTL daemon configuration on one of nodes look like this:

```
portal-group pg1A {
        tag 0x0001
        listen 10.20.20.234:3260
        foreign
}
portal-group pg1B {
        tag 0x8001
        listen 10.20.20.235:3260
}

lun "aluademo" {
        ctl-lun 0
        serial "ac1f6b0c248600"
        device-id "iSCSI Disk      ac1f6b0c248600                  "
        option vendor "TrueNAS"
        option product "iSCSI Disk"
        option revision "0123"
        option naa 0x6589cfc000000d7a21ae0e095faf3cea
}

target iqn.2005-08.com.ixsystems:aluademo {
        alias "aluademo"
        portal-group pg1A no-authentication
        portal-group pg1B no-authentication

        lun 0 "aluademo"
}
```



You may see two iSCSI portals configured with IPs of different controllers. Different portal group tags mean those portal groups represent separate SCSI ports and cannot share MC/S session connections. You may see a single SCSI target associated with those portal groups. And you may also see a single Logical Unit associated with that target, including a set of different unique IDs required for multipath operation.

Now we connect to this target using the FreeBSD iSCSI initiator:

```
# iscsictl -A -d 10.20.20.234
# iscsictl -A -d 10.20.20.235
# iscsictl
Target name                                Target portal     State
iqn.2005-08.com.ixsystems:aluademo         10.20.20.234      Connected: da0
iqn.2005-08.com.ixsystems:aluademo         10.20.20.235      Connected: da1
```

Since the FreeBSD initiator does not support peripheral driver-level multipathing, the two paths to one Logical Unit were detected as two separate block devices, `da0` and `da1`. Looking now at the storage side, we see lists of initiator and target ports inside CTL:

```
# ctladm portlist -i
Port Online Frontend Name      pp vp
0    NO     ha        1:camsim 0  0  naa.5000000341ab1b01
  Target: naa.5000000341ab1b00
1    YES    ha        1:ioctl  0  0
2    YES    ha        1:tpc    0  0
3    YES    ha        1:iscsi  1  1  iqn.2005-08.com.ixsystems:aluademo,t,0x0001
  Target: iqn.2005-08.com.ixsystems:aluademo
  Initiator 0: iqn.1994-09.org.freebsd:mini.ixsystems.com,i,0x8047c8217cb4
128  NO     camsim   camsim   0  0  naa.50000003530efb81
  Target: naa.50000003530efb00
129  YES    ioctl    ioctl    0  0
130  YES    tpc      tpc      0  0
131  YES    iscsi    iscsi    32769 1  iqn.2005-08.com.ixsystems:aluademo,t,0x8001
  Target: iqn.2005-08.com.ixsystems:aluademo
  Initiator 0: iqn.1994-09.org.freebsd:mini.ixsystems.com,i,0x808be26435c8
```

Here we see two groups of target ports, serviced by different storage controllers: ports 0-127 belonging to controller «a», ports 128-255 belonging to controller «b». Controller «b» is the one with ZFS pool access at this point. You see each controller with its own iSCSI target port with unique name. Each of the iSCSI target ports has one connected initiator port, belonging to the same initiator system, but different session IDs (not part of MC/S or a reconnection attempt).

This is where the multipath client operation starts. Using the `sg_inq` tool from the `sg3_utils` package, we fetch identification information for both block devices `da0` and `da1` reported by FreeBSD:

```
# sg_inq -p di da0
VPD INQUIRY: Device Identification page
  Designation descriptor number 1, descriptor length: 59
    designator_type: T10 vendor identification,  code_set: ASCII
    associated with the Addressed logical unit
      vendor id: TrueNAS
      vendor specific: iSCSI Disk      ac1f6b0c248600
  Designation descriptor number 2, descriptor length: 20
    designator_type: NAA,  code_set: Binary
```

list continues

continued from page 7

```
    associated with the Addressed logical unit
      NAA 6, IEEE Company_id: 0x589cfc
      Vendor Specific Identifier: 0xd7a
      Vendor Specific Identifier Extension: 0x21ae0e095faf3cea
      [0x6589cfc000000d7a21ae0e095faf3cea]
  Designation descriptor number 3, descriptor length: 48
    transport: Internet SCSI (iSCSI)
    designator_type: SCSI name string,  code_set: UTF-8
    associated with the Target port
      SCSI name string:
      iqn.2005-08.com.ixsystems:aluademo,t,0x0001
  Designation descriptor number 4, descriptor length: 8
    transport: Internet SCSI (iSCSI)
    designator_type: Relative target port,  code_set: Binary
    associated with the Target port
      Relative target port: 0x3
  Designation descriptor number 5, descriptor length: 8
    transport: Internet SCSI (iSCSI)
    designator_type: Target port group,  code_set: Binary
    associated with the Target port
      Target port group: 0x2
  Designation descriptor number 6, descriptor length: 40
    transport: Internet SCSI (iSCSI)
    designator_type: SCSI name string,  code_set: UTF-8
    associated with the Target device that contains addressed lu
      SCSI name string:
      iqn.2005-08.com.ixsystems:aluademo

# sg_inq -p di da1
VPD INQUIRY: Device Identification page
  Designation descriptor number 1, descriptor length: 59
    designator_type: T10 vendor identification,  code_set: ASCII
    associated with the Addressed logical unit
      vendor id: TrueNAS
      vendor specific: iSCSI Disk      ac1f6b0c248600
  Designation descriptor number 2, descriptor length: 20
    designator_type: NAA,  code_set: Binary
    associated with the Addressed logical unit
      NAA 6, IEEE Company_id: 0x589cfc
      Vendor Specific Identifier: 0xd7a
      Vendor Specific Identifier Extension: 0x21ae0e095faf3cea
      [0x6589cfc000000d7a21ae0e095faf3cea]
  Designation descriptor number 3, descriptor length: 48
    transport: Internet SCSI (iSCSI)
    designator_type: SCSI name string,  code_set: UTF-8
    associated with the Target port
      SCSI name string:
      iqn.2005-08.com.ixsystems:aluademo,t,0x8001
  Designation descriptor number 4, descriptor length: 8
    transport: Internet SCSI (iSCSI)
    designator_type: Relative target port,  code_set: Binary
    associated with the Target port
      Relative target port: 0x83
  Designation descriptor number 5, descriptor length: 8
    transport: Internet SCSI (iSCSI)
```

continued from page 8

```
    designator_type: Target port group,  code_set: Binary
    associated with the Target port
      Target port group: 0x3
  Designation descriptor number 6, descriptor length: 40
    transport: Internet SCSI (iSCSI)
    designator_type: SCSI name string,  code_set: UTF-8
    associated with the Target device that contains addressed lu
      SCSI name string:
      iqn.2005-08.com.ixsystems:aluademo
```

Comparing the outputs, we see that all reported descriptors associated with the addressed Logical Unit are identical. It means those two devices, indeed, represent the same logical unit. Descriptors associated with the target port are different and provide full information to identify them within the target device. You'll see the numbers there match the information reported by the `ctladm portlist` command.

And now, ALUA appears. The primary SCSI command for fetching ports statuses is `REPORT TARGET PORT GROUPS:`:

```
# sg_rtpg da0
Report target port groups:
  target port group id : 0x2 , Pref=0, Rtpg_fmt=0
    target port group asymmetric access state : 0x01
    T_SUP : 1, O_SUP : 0, LBD_SUP : 0, U_SUP : 1, S_SUP : 1, AN_SUP : 1, AO_SUP : 1
    status code : 0x02
    vendor unique status : 0x00
    target port count : 03
    Relative target port ids:
      0x01
      0x02
      0x03
  target port group id : 0x3 , Pref=0, Rtpg_fmt=0
    target port group asymmetric access state : 0x00
    T_SUP : 1, O_SUP : 0, LBD_SUP : 0, U_SUP : 1, S_SUP : 1, AN_SUP : 1, AO_SUP : 1
    status code : 0x02
    vendor unique status : 0x00
    target port count : 03
    Relative target port ids:
      0x81
      0x82
      0x83
```

Here we see that our target reports two port groups (one per storage controller) and 6 ports (3 per storage controller). Groups and ports identifications here match earlier outputs of the `sg_inq` and `ctladm portlist` commands. In addition to information we already know, this command tells us that each group supports a number of ALUA states, and also that port group 0x2 (controller «a») is now in «Active/Non-optimized» state (0x01), while port group 0x3 (controller «b») is in «Active/Optimized» state (0x00). It tells us that at this moment it is preferable to send all requests via the block device `da1`, but if we lose connectity via that path, the `da0` device can also handle requests, just more slowly.

Unfortunately, the FreeBSD SCSI disk peripheral driver does not support multipathing for using that information automatically. The multipathing setup on the GEOM layer has to be done either manually by the administrator, or by some external scripts, as is done, for example, by FreeNAS software for multipath SAS disks. Here we do it manually:

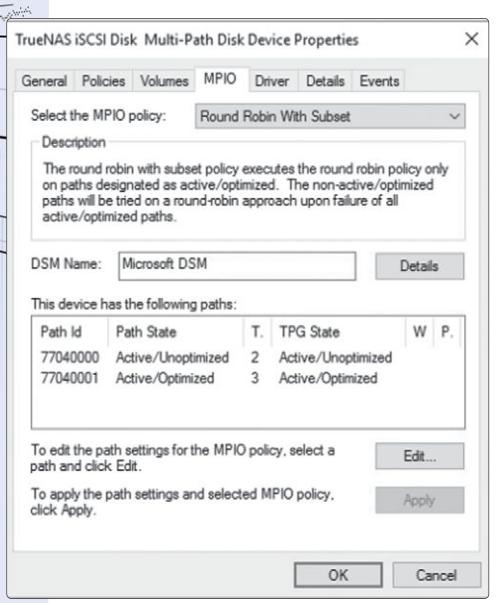
see next page

```
# kldload geom_multipath
# gmultipath label mp0 da0 da1
# gmultipath prefer mp0 da1
# gmultipath status
          Name    Status  Components
multipath/mp0   OPTIMAL  da0 (PASSIVE)
                         da1 (ACTIVE)
```

We see that both da0 and da1 devices do now belong as paths to the multipath/mp0 device, and the da1 device will handle the I/O until either some error occurs or the administrator commands otherwise. Some other initiators may do it completely automatically. For example, VMware vSphere automatically detects multipath devices and configures their operation using ALUA.

Here the initiator correctly identified that two paths belong to the same target, and send I/Os via the proper controller «b». If there is more than one path via that controller, it would be possible to use either the Most Recently Used (default) or the Round Robin path selection policy. In the case of the Round Robin policy, vSphere switches the active path after a certain amount of requests (1,000 by default). The fairly high default value does not allow complete throughput utilization of all paths with only one server, but, as described above, it maintains the original request ordering.

Out of the box, Windows Server only supports MC/S transport-layer multipathing. The upper-layer multipathing has to be installed as a separate, optional feature. But after the component installation and couple system reboot, if your system is still alive ;), the ALUA multipath target should be properly detected and work:



To summarize: used as a target, FreeBSD CAM Target Layer (CTL) can provide decent SCSI multipathing functionality, supporting ALUA and High-Availability clustering, and compatible with many third-party intiators. Transport layer multipathing in the case of iSCSI MC/S is not supported due to high complexity and limited scope, but it could possibly be useful in environments with many Windows desktops. Being used as the initiator, FreeBSD, at the moment, can propose only basic GEOM layer multipathing without ALUA support. To some degree, this can be compensated for by external scripting, but it is prudent to implement full SCSI layer multipathing with ALUA support right out of the box.

ALEXANDER MOTIN is Team Lead of the OS/Services team at iXsystems Inc., and has been a FreeBSD source committer since 2007.

BSD CERTIFICATION GROUP HAS JOINED WITH LPI.

ONE MORE STEP TOWARDS A

FREE AND OPEN SOURCE WORLD

NOT TO MENTION, JOBS!

Now as a part of LPI, BSD certification will gain a new global reach. It will also benefit as it's relaunched in 2018 to fit into a broader program of free and open source professional credentials. You can help by participating in retooling the certification at lpi.org/bsd.

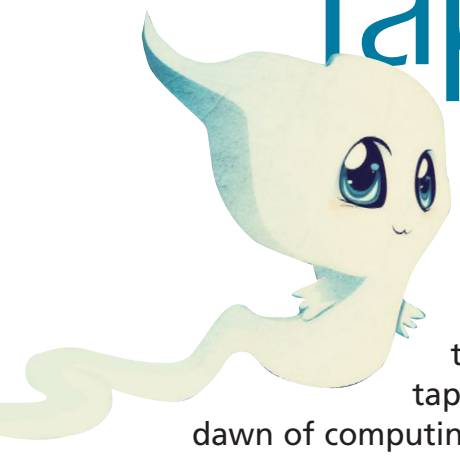
Linux Professional Institute

COMING IN 2018. WATCH FOR UPDATES.

SEE TEXT ONLY

Tape's Not Dead

BY KEN MERRY



For many people evaluating storage alternatives and surveying the landscape of NAS, SAN, Object, Cloud, disk and flash storage, tape is perhaps an anachronism. Tape has been around since the dawn of computing, but does it have a place now? I believe it does. I spent more than 10 years of my career working for two start-ups that explicitly aimed to replace tape with (mostly FreeBSD-based in those cases) disk. I now work for a company that sells a lot of tape products as well as (FreeBSD-based) products that connect to disk and tape. So, I have seen both sides of the debate.

Where Does Tape Fit?

There are places where tape makes a lot of sense, and places where it doesn't. In the 1980s and 1990s, QIC and DAT technologies were aimed at home and smaller business users, and worked well enough for end-user backups. There is no equivalent small, end-user tape technology today. If you're a home user with a couple of gigabytes or maybe even a couple of terabytes to back up, tape isn't generally what you should be looking for.

Before we talk about where tape makes sense, though, it would be helpful to review some recent tape drives and cartridges, and their capabilities:

Drive	Media Type	Raw Capacity	Compressed	Raw Speed	Comp. Speed
IBM TS1155	JD	15TB	37.5TB	360MiB/sec	750MiB/sec
IBM TS1150	JD	10TB	25TB	360MiB/sec	750MiB/sec
LTO-8	LTO-8	12TB	30TB	360MiB/sec	750MiB/sec
LTO-8	LTO-M8	9TB	22.5TB	300MiB/sec	700MiB/sec
LTO-7	LTO-7	6TB	15TB	300MiB/sec	700MiB/sec
Oracle T10000D	T2	8.5TB	21.25	252MiB/sec	800MiB/sec

- The costs of the above drives and media vary widely, especially when you consider that most drives will be in a tape library, and the tape libraries will usually have multiple drives. The more data you have, the more tape makes sense. At large scale, the cost of the media is generally larger than the library and drives.
- Tape provides a number of advantages over disk and flash for storage:
- The cost per GB of tape is very low, generally lower than the cheapest disk-based backup.
- Tape makes a great archive storage media. If you have data that isn't used very often, you can store it on tape and save your expensive disk or flash arrays for frequently used data.
- The throughput is higher than most spinning disk drives. (Although at any scale, you have to consider the total number of tape or disk drives and the interconnect characteristics to evaluate the overall system throughput.)
- Tapes are designed to be moved around and stored. Disks aren't generally designed with that in mind. (Although they can be used that way.) This makes them great for off-site backup.
- Tapes generally have a 30-year shelf life. You won't find disk manufacturers making that claim.
- Tapes are good for air-gapped storage. Are you worried that a software bug, a malicious person, or ransomware will destroy your data? Back it up on tape and put it in off-site storage. Those threats can only destroy data that they can access.

Tapes provide a certain amount of genetic

diversity in storage. What would happen if you had a disk or flash drive firmware bug that corrupted data on all of your drives? What would happen if you had a filesystem bug that corrupted all of your data? With tape, you have different firmware, different media, and different software storing the data. As long as you take the time to back up, you can recover from a drive firmware bug or a filesystem or OS bug.

Tape is not a good fit for data that needs to be accessed very quickly (e.g., less than 10 seconds) or randomly. For those types of data, disk and flash are a better choice.

Tape in FreeBSD

FreeBSD has had tape support since it was the 386BSD patchkit. The original FreeBSD SCSI layer was written by Julian Elischer, who also wrote the st(4) tape driver. Justin Gibbs and I wrote FreeBSD's CAM (Common Access Method) SCSI layer, which went into the FreeBSD tree in 1998 and included the sa(4) (Sequential Access) tape driver. (Justin wrote most of the sa(4) driver.) Matt Jacob rewrote the sa(4) driver substantially starting in late 1998, and much of what it is now is due to his work. Matt was the maintainer of the driver for many years. I am the de-facto maintainer now.

The sa(4) driver supports tape drives that range from old SCSI-1 9-track tape drives to the latest Fibre Channel attached IBM TS1155 tape drives. It includes modern density support, using the mt(1) getdensity subcommand. You can use that command to ask a modern tape drive what kinds of cartridges it supports and in which formats. That is very helpful for IBM TS (and now LTO) tape drives, where one cartridge can have multiple available formats with different capacities.

The sa(4) driver also supports unmapped I/O. Unmapped I/O buffers are userland buffers that are not mapped into the kernel's virtual address space. Instead, they are translated directly to physical pages, which are then transmitted to the FC/SAS/SCSI controllers firmware for DMA. Mapping user buffers into kernel virtual address space produces slowdowns on modern machines with lots of cores due to the TLB (Translation Lookaside Buffer) shootdowns necessary to update the cache in each core. Unmapped I/O allows data that the kernel doesn't need to touch to pass through the kernel without being mapped, avoiding the TLB shootdown. With lots of I/O activity, avoiding TLB shootdowns can boost performance noticeably.

Handling Tape Libraries

FreeBSD has had built-in support for tape libraries via the ch(4) driver and chio(1) utility since the transition from the original SCSI layer to CAM in 1998. I ported the ch(4) driver and chio(1), which were written by Jason Thorpe, from NetBSD to FreeBSD/CAM. The ch(4) driver supports everything from very old libraries to the latest tape libraries.

The mtx(1) utility (in ports/misc) can also control a tape library and operates via SCSI passthrough.

While chio(1) and mtx(1) are good for command line control of a library, a backup application with tape library-level support (like Bacula or Amanda) will use chio(1) or mtx(1) to move tapes between slots and tape drives in a tape library.

Data Integrity

A backup isn't much good if the data gets corrupted; data integrity is very important. You want to be sure that your backups work and that all the bits make it to tape in the right order.

The tape API presents unique challenges for storage protocols like SCSI. When the OS is talking to a hard drive via SCSI, hardware, software, or firmware failures can sometimes cause commands or data to get dropped. If that happens, the driver times out the outstanding command, aborts it, and sends status up to the CAM midlayer so that the command can be retried if the user requested retries. In the case of a timeout on a write, the command and data may or may not have reached the drive originally, but resending the command does no harm. The hard drive will simply rewrite the data again and move on.

Tape is a sequential medium, though. Writes are sent to tape without an explicit logical block address. The address is implicit. Each write (or read) advances the block number down the tape. If a write sent to a tape drive times out, the OS has a dilemma: what made it down to the drive? Did the entire block make it down, and we simply didn't get status back? Did the drive get part of the data? Did the drive get none of the data? Retrying without knowing what made it to the drive could lead to data corruption. The alternative would seem to be aborting the entire backup job with an error.

The ANSI T10 (t10.org) and T11 (t11.org) committees, who write the SCSI and Fibre Channel specifications, respectively, came up with a solution to the problem. It was originally called FC-

Tape, and is now included in the FCP-4 (Fibre Channel Protocol) specification from T10. Sections 4.4 through 4.7 of the spec outline Fibre Channel features needed to assure data integrity for tape drives. They are:

Precise Delivery of Commands

This feature provides a way for an initiator (server) to specify a CRN (Command Reference Number) with each command. The CRN is a number from 1 through 255 that wraps around. If the target (tape drive in this case) receives a command that is not in sequence with the previous command and CRNs are enabled, it will reject the command. This insures that tape I/O requests are executed in order.

Confirmed Completion of FCP I/O Operations

Confirmed completion is a Fibre Channel option that lets the Target (in this case, a tape drive) request that the Initiator (the server) tell the target that it has received a status response. This lets the target know that it does not need to retransmit status to the initiator.

A normal SCSI write command sequence goes like this:

```
Initiator -> Target: I want to write 512KiB.
Target -> Initiator: Go ahead and Transfer the data.
Initiator -> Target: Transferring 512KiB of data.
Target -> Initiator: Data successfully accepted into cache.
```

At this point, the Initiator knows that the Target has completed the write. The Target knows that it has completed the write, but does not know whether the Initiator received the message. Confirmed completion adds another message:

```
Initiator -> Target: I got your completion message.
```

Retransmission of Unsuccessfully Transmitted IUs (Information Units)

If a read or a write command to a Fibre Channel device is taking too long, this feature gives the Fibre Channel driver and/or firmware a mechanism to do link level error recovery without the big hammer of timeouts, aborts, and retries (or rather failures in the case of tape).

If a target (tape drive) takes more than 3 or 4 seconds to respond to a command or a data transmission, the initiator (server) can send REC (Read Exchange Concise) ELS (Extended Link Services) Fibre Channel command to the target to find out the status of the command.

If the target never got the command, the initiator now knows that (as opposed to not knowing what got dropped) and can retransmit it. If the target only got half the data, the initiator now knows that, and can retransmit the data using the SRR (Sequence Retransmission Request) ELS command if the target supports SRR.

This allows link level error recovery in a time frame that is much smaller than typical tape timeouts. The sa(4) driver uses a 32-minute timeout for read and write commands. That is the recommended timeout value from an IBM LTO-5. Other drives are similar. To see the timeout values for your tape drive, try this:

```
camcontrol opcodes sa0 –T
```

So if a problem happens on a link, without FC-Tape, the FC driver will wait 32 minutes, abort the I/O, and then send a failure notification back up the stack to the sa(4) driver. There are no retries, because you can't safely retry tape I/O operations for reasons we've outlined.

With FC-Tape, though, the FC driver or firmware will send a REC to find out the status of the command, and assuming the drive is still responsive, the initiator and target can get the transfer moving again in seconds as opposed to minutes.

This capability can also help ensure reliable tape access in a Fibre Channel environment with reliability issues. An analogue from the network world is using TCP (Transmission Control Protocol) on top of IP (Internet Protocol) to ensure reliable transmissions on a network that can sometimes drop packets.

Task Retry Identification

Task Retry Identification is a mechanism for identifying a retry sequence. This is a necessary addition because of the way commands are identified in Fibre Channel.

In Fibre Channel, the initiator (server) gives an OX_ID (Originator Exchange ID) to each SCSI (or other) command it sends to the target (tape drive). When the target responds to the initial command, it replies with the same OX_ID and adds its own RX_ID (Receiver Exchange ID). The OX_ID and

RX_ID are how a particular command is identified from then on. When data is transmitted to the target, the initiator specifies the OX_ID and RX_ID so that the target knows which command the data belongs to.

The OX_ID and RX_ID are 16 bite values can quickly wrap around. For this reason, the Task Retry Identifier is specified along with Fibre Channel commands (FCP_CMND), REC, and SRR to tie the error recovery actions together and make it clear which command they're referring to.

FreeBSD FC-Tape Support

FreeBSD has three Fibre Channel drivers, two of which are in the tree and one that will be in the tree soon.

The isp(4) driver was written by Matt Jacob and covers Qlogic controllers from Parallel SCSI all the way to 16Gb Fibre Channel. Matt added FC-Tape support in 2012 thanks to a contract from Spectra Logic Corporation.

The mpt(4) driver includes support for older LSI FC controllers up to 4Gb, but does not support FC-Tape.

The ocs_fc(4) driver, which was written by Broadcom (formerly Emulex), should be committed to FreeBSD soon. It supports Broadcom's 16Gb and 32Gb Fibre Channel cards and does support FC-Tape.

SAS

Fibre Channel includes a number of link level error recovery features for tape drives, but what about SAS?

SAS includes a feature called TLR (Transport Layer Retries) that is similar to the FC-Tape features. It can retransmit commands and data as needed. For more details about how this works, get the SPL-4r12 (SAS Protocol Layer 4, revision 12) specification from t10.org and look in section 8.2.1.

The mps(4) (LSI/Broadcom 6Gb SAS) and mpr(4) (LSI/Broadcom 12Gb SAS) drivers in FreeBSD support TLR and turn it on for tape drives.

Bit Error Rate and Checksums

Another topic that comes up when talking about hard drives and tape drives is Bit Error Rate. The Bit Error Rate is the likelihood of a tape or disk drive returning a byte to the host that is incorrect. You can also think of it as the likelihood of silent data corruption.

If your data is corrupt on the disk or tape, you want the drive to tell you about it, and not pass back bad data.

Disk and tape drives use various algorithms to reduce the likelihood of silent data corruption, and, by default, the tape algorithms are better. But, you can add checksums to both disk and tape storage to significantly reduce the likelihood of silent data corruption. (Using ZFS is an excellent way to reduce the incidence of silent data corruption with disk.)

The Bit Error Rate topic alone could fill a fairly lengthy article, so instead of doing that here, this article covers disk and disk channel bit error rate pretty well:

http://www.enterprisestorageforum.com/storage-technology/sas-vs.-sata-1.html

And this paper from the LTO consortium briefly explains that tape drives have better error detection and correction algorithms and so are much less prone than disks to silent data corruption:

https://www.lto.org/wp-content/uploads/2014/06/LTO16_0026_ValueProp_Reliability_01_2016_FINAL.pdf

Modern tape drives also support Protection Information, which is an extra CRC (CRC32 or Reed-Solomon CRC) that can be written with each tape block, and checked by the tape drive on read and write. The FreeBSD sa(4) driver supports Protection Information. The only application I know of that supports adding the CRC to each block on FreeBSD is IBM's LTFS. (See below.) See the mt(1) man page section on the 'protect' subcommand and the sa(4) man page for more details on how it works.

Users and applications are also free to write their own checksums to tape and read them back. Even if tape has a Bit Error Rate that is much better than disk, that doesn't help if the data gets corrupted before it makes it to the tape, or after it is read off of the tape. (Protection Information, above, can help in that situation, especially if it is generated in the application.)

Application Support

There are many applications that talk to tape, but there are two major open-source backup applications that run on FreeBSD and talk to tape: Amanda and Bacula. There are also other built-in tools you can use to talk to tape. And finally, there is LTFS.

Amanda

Amanda stands for Advanced Maryland Automatic Network Disk Archiver. The Amanda home page is at http://amanda.org. The Amanda server is in the ports tree at misc/amanda-server.

Amanda offers a myriad of features. One of the things I like about Amanda is that it works with ZFS snapshots. When you do a full backup, it creates a new snapshot of the ZFS filesystem and uses 'zfs send' to send the snapshot to tape. If you do an incremental backup, though, it creates another snapshot and does an incremental 'zfs send' to capture the changes from the last Amanda snapshot.

The drawback to backing up ZFS snapshots is that if you have to restore a single file, you will have to pull the entire snapshot for the filesystem in question off tape in order to restore the file. If the file is on an incremental backup, you'll have to restore the incremental and the full snapshot from tape to restore the file.

You can reduce the chances of needing to restore an individual file by establishing periodic ZFS snapshot creation. If you have hourly, daily, weekly, and monthly snapshots (pruned accordingly so you don't keep too much history), users can restore accidentally deleted files themselves. This is much faster than pulling the file off tape and hopefully requires no administrator support other than pointing the user in the right direction.

Bacula

Bacula is the other major open-source backup package available on FreeBSD. It offers a myriad of features as well. It is available in the ports tree at sysutils/bacula-server.

One of the major features of Bacula is that it does do file-based backup. If you do need to restore individual files from tape, and you don't want to pull the entire filesystem off tape, Bacula is an excellent solution. It stores the list of files in a database like Postgres or MySQL.

Built-in Tools

FreeBSD ships with several tools that natively talk to tape.

- tar(1) is not just a utility for distributing source. It stands for Tape ARchive, and tar variants have been around since UNIX Version 7. If you need to quickly send a group of files to tape, tar(1) will do that for you, and the tape should be readable on most any Unix system.
- dump(8) is the system utility for backing up UFS filesystems, and it also talks to tape. Dump will preserve the full metadata of the filesystem, and is a great tool for full backups. You can also do incremental backups with dump(8).
- dd(1) also knows how to talk to tape. If you just want to copy blocks off a tape, you can use dd(1) to do it. For example: "dd if=/dev/nsa0 of=my_tape_image bs=128k". dd will read until it encounters a file mark. You can run dd again to pull blocks off tape after you encounter a filemark.
- camdd(8) knows how to talk to tape. I wrote it as a faster, multithreaded version of dd and as an example of how to use to the asynchronous pass(4) driver interface. One of its features, though, is that it can talk to tape. For example: "camdd –i pass=da5,bs=128k,depth=8 –o file=/dev/nsa0,bs=128k". That will issue 8 128KiB reads at a time via the pass(4) driver to the disk da(5) and write those blocks in order (which is very important!) to tape one at a time.

LTFS

LTFS stands for the Linear Tape File System. This is what it seems to be—a filesystem on a tape. It is intended to be both a filesystem that you can use to read and write individual files and a standard interchange format.

LTFS uses two partitions on tape, an Index partition and a Data partition. All of the filesystem metadata, including the location of each file's blocks on tape, is stored in an XML file that lives on the index partition and is included in the data partition. When you add files to the filesystem, a new version of the index is generated and written to tape.

IBM originally wrote LTFS, and transferred the standard to SNIA (https://www.snia.org/tech_activities/standards/curr_standards/ltfs). The LTO Consortium (http://lto.org) offers compliance testing and certification for LTFS implementations, so that different vendors can insure their implementations will work with others.

IBM has offered the source for its version of LTFS (also called Spectrum Archive Single Drive Edition) that talks to IBM tape drives for a number of years. IBM BSD licensed their version of LTFS in October 2017 (it was previously licensed under the LGPL), and it is available here:

https://github.com/LinearTapeFileSystem/ltfs

I ported IBM's LTFS to FreeBSD in 2013 (sponsored by Spectra Logic), and I am in the process of preparing that work for a pull request so that it can go into IBM's tree.

LTFS works with LTO-5 drives and higher. It requires tape partition support and a big enough flash chip on the tape (called the MAM—Media Auxiliary Memory).

IBM's LTFS uses FUSE (Filesystem in Userspace) to present a file interface. LTFS itself runs as a userland process, and FUSE forwards the VFS requests to the LTFS process.

LTFS isn't a replacement for a backup program like Amanda or Bacula. That is because it only talks to individual tapes. It's left to the user (or vendor) to wrap a larger application around it to manage tapes in the tape library and move data to and from the tape.

The LTFS filesystem on FreeBSD would mostly be useful as a tape interchange format. (Many media companies use it.) If you have a lot of data, you would really want a bigger application that could use LTFS as its on-tape format.

Conclusion

Tape is still alive and is actively developed and used for storing large volumes of data. Even if your data lives in the cloud, it is likely backed up on tape.

If you have large volumes of data to back up, you need air-gapped or offsite storage, or you just want some genetic diversity in your storage, tape may be a good solution. ●

KEN MERRY has been a FreeBSD committer since 1998, and a FreeBSD user since 1994. He is the coauthor of the *FreeBSD CAM I/O subsystem* and the author of *CTL, the CAM Target Layer*. He lives near Atlanta with his wife and two sons.

SEE TEXT ONLY

TRACING `ifconfig` Commands from Userspace to Device Driver

I am currently working on expanding FreeBSD's `rtwn(4)` wireless device driver. I have the basics down, such as initialization, powering on and off, loading the firmware, etc., and am now trying to fill in specific `ifconfig(8)` methods. This requires an in-depth knowledge of how `ifconfig(8)` commands are ultimately delivered to the driver. I could not find concise documentation that outlines each stage of the process, so I wrote one!

BY FARHAN KHAN

This article is specific to FreeBSD 12.0-CURRENT, but it should apply to any future version and other operating systems that utilize `net80211(4)`, such as OpenBSD, NetBSD, DragonFlyBSD, and illumos. I hope it serves to help the FreeBSD community continue to develop WiFi and other device drivers. This is not an exhaustive guide, but it should provide you with the basic order of operations. In this example, I walk through changing the channel on your WiFi card and placing it in monitor mode as follows:

```
# ifconfig wlan0 channel 6 wlanmode monitor
```

High-level Summary

FreeBSD's `ifconfig(8)` utilizes the `lib80211(3)` userspace library which functions as an API to populate kernel data structures and issue the `ioctl(2)` syscall. The kernel receives the `ioctl(2)` syscall in a new thread, interprets the structure, and routes the command to the appropriate stack. In our case this is `net80211(4)`. The kernel then creates a new queued task and terminates the thread. Later on, a different kernel thread receives the queued task and runs the associated `net80211(4)` handler, which immediately delivers execution to the device driver.

To summarize again:

Userspace: `ifconfig(8)` binary
Userspace: `lib80211(3)` library
Kernel: `net80211(4)` code
Kernel: `taskqueue(9)` to the driver
Kernel: Device Driver

Let's begin!

Userspace: `ifconfig(8)` + `lib80211(3)` library

Early in `ifconfig(8)`, it opens a `SOCK_DGRAM` socket in `/usr/src/sbin/ifconfig/ifconfig.c` as follows:

```
s = socket(AF_LOCAL, SOCK_DGRAM, 0).
```

This socket functions as the interface for userspace to kernel communication. Rather than tracing from the `if-else` maze in `main()`, I grepped for the string "channel", and found it in `ieee80211_cmd[]`defined at the end of `/usr/src/sbin/ifconfig/ifieee80211.c`.

This table enumerates all `ieee80211` `ifconfig(8)` commands. The "channel" command is defined as follows:

```
DEF_CMD_ARG ("channel" set80211channel).
```

Note the second argument. I looked up `DEF_CMD_ARG` and found that it was a pre-processor macro that defines what function is run when the user sends `ifconfig(8)` a command. A quick grep search shows `set80211channel` is defined in `/usr/src/sbin/ifconfig/ifieee80211.c`. The parameters are fairly easy to identify: `val` is the new channel number (1 through 14) and `s` is the socket we opened earlier. This executes `ifconfig(8)`'s `set80211` function whose sole purpose is to cleanly transfer execution into the `lib80211(3)` library.

`lib80211(3)` is an 802.11 wireless network management library to formally communicate with the kernel. It is worth noting that neither OpenBSD nor NetBSD has this library and instead opt to communicate directly to the kernel. As mentioned, `ifconfig(8)`'s `set80211` function calls `lib80211_set80211` located in `/usr/src/lib/lib80211/lib80211_ioctl.c`. The `lib80211_set80211` function populates an `ieee80211reqdata` structure, used for user-to-kernel `ieee80211` communication. In the example below, this is the `ireq` variable, which contains the WiFi interface name and intended channel.

The library then calls the ioctl(2), as follows:

```
ioctl(s, SIOCS80211, &ireq).
```

This runs the syscall to formally enter kernel-space execution. In essence, `ifconfig(8)` is nothing more than a fancy `ioctl(2)` controller. You could write your own interface configuration tool that directly calls the `ioctl(2)` syscall and get the same result. Now on to the kernel!

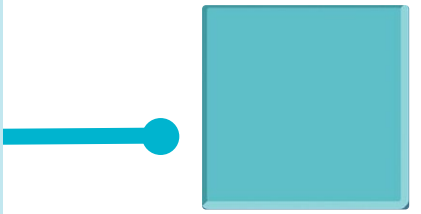
Kernel Command Routing to `net80211(4)`

There are two brief explanations before we proceed: First, at a high level, the BSD kernel operates like an IP router in that it routes execution through the kernel, populating relevant data values along the way until the execution reaches its destination handling functions. The following explanation shows how the kernel will identify the syscall type, determine that it is for an interface card, determine the type of interface card, and finally queue a task for future execution.

Second, the BSD kernel utilizes a common pattern of using template methods that call a series of function pointers. The exact function pointers are conditionally populated, allowing the code to maintain a consistent structure while the exact implementation may differ. It works very well, but can make tracing execution paths difficult if you are just reading the code straight through. When I had trouble, I typically used illumos's OpenGrok (http://src.illumos.org/source/) or dtrace(1).

Let's take a brief detour into dtrace(1). Solaris's dynamic tracing tool is imported to FreeBSD and used to monitor a kernel or process in real time. It is useful in understanding what the operating system is doing and saves you the trouble of using printf(3)-style debugging. I used dtrace(1) in writing this guide to identify what the kernel was executing, function arguments, and the stack trace at any given moment. For example, if I wanted to monitor the ifioctl function, I might run this:

```
# dtrace -n '
> fbt:kernel:ifioctl:entry {
> self->cmd = args[1];
> stack(10);
> }
> fbt:kernel:ifioctl:return {
> printf("ifioctl(cmd=%x) = %x", self->cmd, arg1);
> exit(0);
> } '
```



This dtrace(1) one-line command sets up handlers for `ifioctl`'s entry and return probes. On entry, dtrace(1) records the value of the second argument `cmd`, and displays the last 10 elements of the stack. On return, it displays the function argument and return value. I used variations of this basic command template throughout my research, especially when I was confused in tracing the code or could not identify a function's arguments.

The first non-assembly function is the amd64-specific syscall handler, `amd64_syscall`, that receives a new thread structure and identifies the type as a syscall. In our case, it is for an `ioctl(2)`, so amd64_syscall calls `sys_ioctl` located in `/usr/src/sys/kern/sys_generic.c`.

On FreeBSD, `sys_ioctl` performs input validation and formats the data it receives. It then calls `kern_ioctl`, which determines what type of file descriptor the `ioctl(2)` is working with, the capabilities for the socket, and assigns the function pointer `fo_ioctl` accordingly. (NetBSD and OpenBSD do not have `kern_ioctl`. For them, `sys_ioctl` directly calls `fo_ioctl`.) Our file descriptor corresponds to an interface, so FreeBSD assigns `fo_ioctl` as a function pointer to `ifioctl`, which handles interface-layer `ioctl(2)` calls. This function is located in `/usr/src/sys/net/if.c`.

The function `ifioctl` is responsible for all sorts of interfaces: Ethernet, WiFi, epair(4), etc. `ifioctl` starts with a switch-condition based on the `cmd` argument. This checks if the command can be handled by

net80211(4) without needing to jump into the driver, such as creating a clone interface or updating the MTU. A quick dtrace(2) probe reveals that the cmd argument is SIOCS80211, which fails to meet any switch-conditions, so execution jumps to the bottom. The function continues and calls ifp->if_ioctl, which, in the case of WiFi, is a function pointer to ieee80211_ioctl, located in /usr/src/sys/net80211/ieee80211_ioctl.c.

ieee80211_ioctl contains another switch-case. With cmd set to SIOCS80211, execution matches the associated case and calls ieee80211_ioctl_set80211, located in /usr/src/sys/net80211/ieee80211_ioctl.c.

ieee80211_ioctl_set80211 has yet another switch-case with a few dozen conditions. The ireq->i_type was set to IEEE80211_IOC_CHANNEL by lib80211(3) so it will match the associated case and execute ieee80211_ioctl_setchannel. The gist of this function is to determine if the input channel is valid, or if the kernel needs to set any other values. It concludes by calling setcurchan, which does two things. First, it determines the validity of the channel and if any additional values must be set. Second, it runs ieee80211_runtask, which makes the final thread-level call to taskqueue_enqueue.

The Kernel: Task Execution

taskqueue_enqueue is not an ieee80211(9) function, but it's worth a brief review. In a nutshell, the taskqueue(9) framework allows you to defer code execution into the future. For example, if you want to delay execution for 3 seconds, running the kernel equivalent of sleep(3) would cause the entire CPU core to halt for 3 seconds. This is unacceptable. Instead, taskqueue(9) allows you to specify a function that the kernel will execute at a later time.

In our channel change example, the scheduled function is the net80211(4) function update_channel, located in /usr/src/sys/net80211/ieee80211_proto.c. When taskqueue(9) reaches our enqueued task, it will first initiate the update_channel handler to receive the task and immediately hand over execution to the driver code pointed to by ic_set_channel.

To summarize, up to this point, the kernel has routed the command to the network stack, which routed to the WiFi-specific stack, where it was scheduled as a task for future execution. When taskqueue(9) reaches the task, it immediately jumps to the driver-specific code. At last, we entered the driver!

The Driver

From here on, the code is driver-specific and I will not get into the implementation details, as each device has its own unique channel changing process. I am currently working on rtwn(9), which is located in /usr/src/sys/dev/rtwn. NetBSD and OpenBSD separate USB and PCI drivers, so the same driver is located in /usr/src/sys/dev/usb/if_urtwn.c and /usr/src/sys/dev/pci/if_rtwn.c, respectively.

Operating systems need a standard way to communicate with device drivers. Typically, the driver provides a structure containing a series of function pointers to driver-specific code, and the kernel uses this as an entry point into the driver code. In the case of WiFi, this structure is ieee80211com, located in /usr/src/sys/net80211/ieee80211_var.h. By convention, all BSD-derived systems use the variable name ic to handle ieee80211(9) methods.

In our case, we are changing the channel, so the operating system will call ic->ic_set_channel, which is a pointer to the driver's channel changing function. For rtwn(9), this is rtwn_set_channel, which itself is a function pointer to r92c_set_chan, r92e_set_chan or r12a_set_chan, depending on which specific device you are using.

The specifics of rtwn(9) are outside of the scope of this article, but it is worth discussing how the driver communicates to the hardware. The softc structure is a struct that maintains the device's runtime variables, states, and method implementations. By convention, each driver's softc instance is called sc. You might wonder why you need yet another method function pointer when ieee80211com provides that. This is because ieee80211com's methods point to command handlers, not necessarily to device routines. Device drivers may have their own internal methods that are not part of ieee80211com. Also, the softcstructure can handle minor variations between device versions. rtwn(9)'s softc struct is

called `rtwn_softc` and is located in `/usr/src/sys/dev/rtwn/if_rtwnvar.h`.

How does a driver send data to the driver? `rtwn(9)` uses the `rtwn_write_[1|2|4]` and `rtwn_read_[1|2|4]` methods to actually send or receive a byte, word, or double-word. `rtwn_read_1` is a pointer to the `sc_read_1` method. The driver assigns the `sc_read` class of functions at initialization to either the `rtwn_usb_read_*` and `rtwn_usb_write_*` methods or `rtwn_pci_read_*` and `rtwn_pci_write_*`. The aforementioned classes of functions are abstractions to the PCI and USB buses. In the case of PCI, these function calls will eventually call `bus_space_read_*` and `bus_space_write_*`, which are part of the PCI subsystem. In the case of USB, the driver will call `usbd_do_request_flags`, which is part of the USB subsystem. A well-written driver should abstract these bus-specific layers and provide you with clean read and write methods for various data sizes. As an aside, FreeBSD is long overdue for an SDIO stack, and this is a major impediment for the Raspberry Pi, Chromebooks, and other embedded devices. But I digress…

As an example, the driver uses the following line to enable hardware:

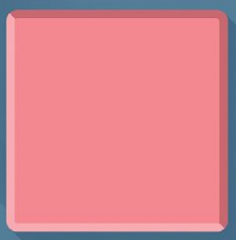
```
interrupts.rtwn_write_4(sc, R92C_HIMR, R92C_INT_ENABLE).
```

This will write the value `R92C_INT_ENABLE` to the `R92C_HIMR` device register.

Conclusion

To summarize this journey, the `ifconfig(8)` opens a socket and passes it to the `lib80211(3)` library. `lib80211(3)` sends a `userspace-to-kernel` command structure to the kernel with an `ioctl(2)` syscall. The syscall triggers the kernel to run a new kernel thread. From here, the kernel determines that the `ioctl(2)` command corresponds to a network card, specifies the type as a WiFi card, and then identifies the exact command type. The `ieee80211(9)` tells `taskqueue` to create a new task to change the WiFi channel, and then terminates. Later on, the `taskqueue(9)` runs the `ieee80211(9)` task handler that transfers execution to the driver. The driver communicates to the hardware using the PCI or USB buses to change the WiFi channel.

In my opinion, FreeBSD is technically superior to Linux, but is still lacking in several critical areas, and among those is hardware support. I hope this article prompts the FreeBSD community to continue to produce high-quality, faster device drivers. ●



FARHAN KHAN is a 2012 graduate from George Mason University in Applied Information Technology Security. He currently works as a Senior Security Engineer for McAfee Security. He considers himself an IPv6 evangelist, avid programmer, and hopes to contribute to the FreeBSD Project.

SEE TEXT ONLY

conference REPORT

by Brooks Davis

BSDTW 2017 held in Taipei, Taiwan, November 11 and 12, 2017, was a great success. I attended as a both a speaker and BSD developer. This was the first BSDTW and the second BSD conference in Taiwan. It was hosted by the Open Culture Foundation and Skymizer, and spearheaded by Li-Wen Hsu.

The conference was smaller than many, with a single track of talks that avoided the all-too-frequent problem of talks I want to attend piling up in one slot. After the opening session, the first day kicked off with a talk on the RISC-V architecture by Arun Thomas. Arun gave us an overview of the architecture, an update on current status, and a preview of upcoming changes. RISC-V seems to be gaining traction in a number of areas, including industry and research, and so it's good that FreeBSD offers solid support for it.

George Neville-Neil was up next with an update on DTrace as used in the CADETS project. They use DTrace in an always-on mode where the traditional DTrace performance guarantees (little or no impact when probes are not enabled) are a poor fit. George covered a number of improvements in progress as well as the work on OpenDTrace, a project to bring various DTrace communities back together along the lines of OpenZFS.

Baptiste Daroussin then discussed work to extend the Poudrière package building system to build FreeBSD images. In some ways, this seems like a divergence in purpose, but in fact, Poudrière has always needed the ability to construct clean FreeBSD images to mount as the OS filesystem when building packages. The new functionality simply adds the ability to bundle those images rather than maintaining them as ZFS volumes.

After lunch I presented my talk on the history of virtual address space management, showing how we progressed from the break() interface in early UNIX to mmap() today. I also discussed why I think it is time to reconsider this interface in light of modern mitigation strategies and notions of API design.

Johannes M. Dieterich followed with a talk on the state of FreeBSD for HPC with a focus on program language support, numerical libraries, and GPU support. As someone who used to work in this area, it was interesting to see how FreeBSD has progressed and where work is needed.

The last talk of the day was given by Peter Grehan, who discussed the challenges of adding graphics support to bhyve. This work was motivated by the fact that without graphics support, there isn't a practical way to install non-server versions of Windows and some Linux distributions. bhyve is now an emulated graphics adaptor, keyboard, and mouse along with a VNC server. This talk was full of interesting examples of how notionally standard things like the VNC protocol are often less standard in practice.

That evening, the conference banquet was held at a restaurant in the Tamsui District. We walked from the end of the train line to the restaurant along the beach where there were many shops selling food, including some amazing-looking fried cephalopods on sticks. Unfortunately, we were in too much of a hurry to get to dinner to have time to partake.

Sunday morning kicked off with Allan Jude talking about advanced ZFS integration. His presentation focused largely on various enhancements to the boot

process, including boot environments, a new next-boot implementation of ZFS, and automated boot recovery. Allan also talked about GELI disk encryption and appliance upgrades using ZFS.

Theo de Raadt was up next with a compelling talk on the use of security mitigations to incrementally improve security and find bug or code quality issues that might become bugs early and fix them. He made a good case of the on-by-default approach taken in OpenBSD. This approach isn't for everyone, but it's excellent that someone is doing it.

Ruslan Bukin delivered a talk on the status of RISC-V support in FreeBSD as well as in the larger software ecosystem. He also walked us through the porting process and discussed changes to support the upcoming v1.10 privilege specification. In-tree Linux support will be based on this version, so it should be close to final.

After lunch Mariusz Zaborski discussed sandboxing applications with Capsicum. Among other things, he detailed the reasons for a variety of services to support existing libc interfaces. A new one (committed around the time of the talk) was the syslog service, which is required because there is no reliable way to pre-open log services in the current API.

The final full talk of the conference was by Mark Johnston on changes to the virtual memory system in FreeBSD. Mark covered a wide range of improvements, from the recent addition of `MAP_GUARD` to prevent some stack clash issues, to improvements to build performance on 128-thread systems that are now available on Amazon.

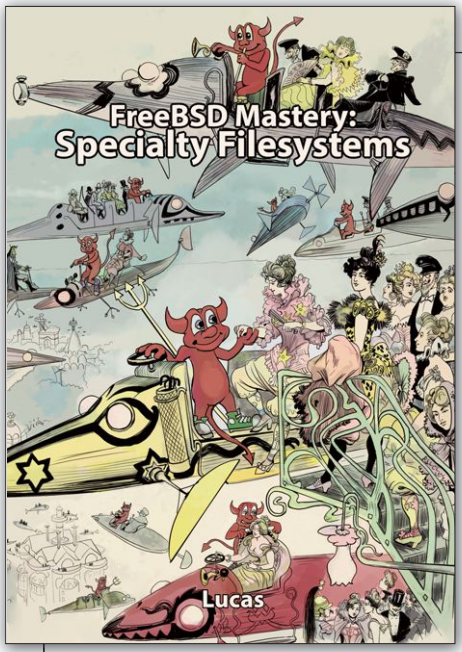
The conference concluded with Lightning Talks where time limits were rigorously enforced. The talks were interesting and the timekeeper kept the speakers on their toes.

I had a little time for tourist activities and enjoyed simply walking the streets of Taipei and a trip up Taipei 101 on a clear day. Unfortunately, there wasn't enough free time in my schedule to visit more of the many interesting sites such as the National Palace Museum. I look forward to future BSDTW conferences and getting to see some of the places I missed this time on my next trip to Taiwan! ●

BROOKS DAVIS is a Senior Software Engineer in the Computer Science Laboratory at SRI International and a Visiting Research Fellow at the University of Cambridge Computer Laboratory. He has been a FreeBSD user since 1994, a FreeBSD committer since 2001, and was a core team member from 2006 to 2012.

SEE TEXT ONLY

BOOKreview by Matt Joras



FreeBSD Mastery: *Specialty Filesystems*

by Michael W Lucas

Publisher	*Tilting Windmill Press (2016)*
Print List Price	*$24.99*
Digital List Price	*$9.99*
ISBN	*9780692610442*
Pages	*238*

It is often difficult to find useful and practical examples for open-source software. Manual pages are all well and good, but they are often most useful once you understand the basics of the software. What's often missing from manual pages is the basic "here's an example of a problem this software solves and how to configure it to do so." Specialty Filesystems by Michael W Lucas fills this void for certain filesystems in FreeBSD.

The book serves as a basic reference, but it is best thought of as a book that showcases some lesser-known features of FreeBSD. It starts off with short, easily consumable examples of how to access various "foreign" media. This section is sprinkled with some interesting historical notes as well.

The next section focuses on devfs(8) and devd(8), both of which I was only vaguely familiar with. In addition to demystifying the purpose and function of devfs(8) and devd(8), Lucas offers useful advice and practical examples that a desktop user of FreeBSD will appreciate. There are additional mentions of procfs(5) and fdescfs(5); these, however, were less illuminating or useful than the sections on devfs(8) and devd(8).

The next set of filesystems covered are the POSIX message queue filesystem, union mounts, null mounts, and memory filesystems. The mqueuefs(5) is a feature I didn't even know was a part of POSIX. It seems that as an inter-process communication method it has fallen out of (or maybe was never in) favor, perhaps without good reason. Its inclusion in the book makes for interesting reading, but such treatment of it may be better for a book focused on programming than systems administration. The union mounts and null mounts sections are useful for those who never thought a filesystem feature existed to solve certain classes of problems. The coverage of the two memory-backed filesystems md(4) and tmpfs(5) provides an overview for someone who needs to understand the differences between them.

The book goes on to give a broad overview of the venerable Network File System (NFS). System administrators who find the distinctions confusing should find this section helpful since the author does a better job at clarifying the situation than what you'll find when Google leads you to questions on Serverfault or Stackoverflow. The same can be said of the handling of Microsoft's CIFs.

There is an interesting and rather in-depth treatment of how to use iSCSI and the HAST feature to create a redundant storage network. This section could almost be used verbatim as a setup guide for someone looking to deploy a similar feature with FreeBSD. I learned a fair bit about the "built in" failover and redundancy features in FreeBSD. Despite its relative specificity compared to other sections of the book, I think it is still valuable as it leverages various "special" features.

The treatment of the FUSE was OK, but did not

add anything spectacularly useful compared to what is readily accessible on the Internet about fusefs. There are some specifics to FreeBSD's FUSE implementation that are legitimately useful, but I do think there are more caveats to the FreeBSD implementation that are not covered and probably should have been.

Personally, the section on automountd(8) was very informative. This is a feature I had seen mentioned here and there when I first started using FreeBSD as a desktop machine, but it was never entirely clear to me what it was doing. Frankly, I thought it was just a basic "see external media, mount it" sort of utility. The book does a great job of showcasing how it can do much more than that. It is definitely an underappreciated feature in FreeBSD that the book does more justice to than I've seen in official documentation and guides.

The section on NFSv4 ACLs did not contribute much overall to the book. It seems like a topic that is better suited to its own text rather than added on to an otherwise easy-to-read technical book.

Overall, the book was an easy and enjoyable read. As a software developer and hobbyist system administrator, I learned a fair bit about FreeBSD's filesystems and gained clarity on many I already knew about. I think the book is a good read for anyone looking to broaden their FreeBSD toolkit to include various filesystem bits.

MATT JORAS is a FreeBSD committer interested in kernel scalability, networking, and FreeBSD desktop/laptop usability. He currently works for Dell EMC developing for the Isilon product, which is based on FreeBSD. At home, he runs FreeBSD on his personal machines and is a hobbyist sysadmin using FreeBSD/ZFS to datahoard dozens of terabytes of personal data.

SEE TEXT ONLY

svn UPDATE

by Steven Kreuzer

What advantages does FreeBSD have over Linux? That's a question I get asked quite a bit, and it is hard to answer because FreeBSD does a lot of things really well. Where do you start? You could ramble on about the robust networking stack or cutting-edge technology such as DTrace and Capsicum for hours on end. However, I would argue that the area where FreeBSD really shines is storage. Not only are developers spending quite a bit of time making sure that the 1s and 0s you save to the disk get written in the correct order, but that it is doing so as quickly as possible. If that wasn't enough, they are also making sure that those same 1s and 0s get read back in the correct order as quickly as possible. Whether you are just archiving family photos on your laptop or serving a multi petabyte ZFS volume to thousands of clients on the network, FreeBSD provides a robust and reliable platform to meet your storage needs.

Provide a sysctl to force synchronous initialization of inode blocks. https://svnweb.freebsd.org/changeset/base/326731

FFS performs asynchronous inode initialization using a barrier write to ensure that the inode block is written before the corresponding cylinder group header update. Some GEOMs do not appear to handle BIO_ORDERED correctly, meaning that the barrier write may not work as intended. The sysctl allows one to work around this problem at the cost of expensive file creation on new filesystems.

Support mounted boot partitions in the installer. https://svnweb.freebsd.org/changeset/base/326674

This allows the platform layer, for example, to specify that the EFI boot partition should be mounted at /efi and formatted normally with newfs msdos rather than splatted to from /boot/boot1.efifat.

zfs_write: fix problem with writes appearing to succeed when over quota. https://svnweb.freebsd.org/changeset/base/326070

The problem happens when the writes have offsets and sizes aligned with a filesystem's recordsize (maximum block size). In this scenario dmu_tx_assign() would fail because of being over the quota, but the uio would already be modified in the code path where we copy data from the uio into a borrowed ARC buffer. That makes an appearance of a partial write, so zfs_write() would return success and the uio would be modified consistently with writing a single block. That bug can result in a data loss because the writes over the quota would appear to succeed while the actual data is being discarded.

This commit fixes the bug by ensuring that the uio is not changed until after all error checks are done. To achieve that, the code now uses uiocopy() + uioskip() as in the original illumos design. We can do that now that uiocopy() has been updated in r326067 to use vn_io_fault_uiomove().

Avoid holding the process in uread() and uwrite(). https://svnweb.freebsd.org/changeset/base/325887

In general, higher-level code will atomatically verify that the process is not exiting and hold the process. In one case, we were using uwrite() to copy a probed instruction to a per-thread scratch space block, but copyout() can be used for this purpose instead; this change effectively reverts r227291.

Optimize telldir(3). https://svnweb.freebsd.org/changeset/base/326640

Currently each call to telldir() requires a malloc and adds an entry to a linked list which must be traversed on future telldir(), seekdir(), closedir(), and readdir() calls. Applications that call telldir() for every directory entry incur O(n^2) behavior in readdir() and O(n) in telldir() and closedir().

This optimization eliminates the malloc() and linked list in most cases by packing the relevant information into a single long representation. On 64-bit architectures msdosfs, NFS, tmpfs, UFS, and

ZFS can all use the packed representation. On 32-bit architectures, msdosfs, NFS, and UFS can use the packed representation, but ZFS and tmpfs can only use it for about the first 128 files per directory. Memory savings is about 50 bytes per telldir(3) call. Speedup for telldir()-heavy directory traversals is about 20-30x for one million files per directory.

Tweak seekdir, telldir, and readdir so that when there are deletes, seeks to the last location saved will work. https://svnweb.freebsd.org/changeset/base/282485

This is needed for Samba to be able to correctly handle delete requests from windows. This does not completely fix seekdir when deletes are present but fixes the worst of the problems. The real solution must involve some changes to the API for eh VFS and getdirentries(2).

Avoid the overhead of acquiring a lock in nfsrv_checkgetattr() when there are no write delegations issued. https://svnweb.freebsd.org/changeset/base/326544

Manu@ reported on the freebsd-current@ mailing list that there was a significant performance hit in nfsrv_checkgetattr() caused by the acquisition/release of a state lock, even when there were no write delegations issued. This patch adds a count of outstanding issued write delegations to the NFSv4 server. This count allows nfsrv_checkgetattr() to return without acquiring any lock when the count is 0, avoiding the performance hit for the case where no write delegations are issued.

zfsd should be able to online an L2ARC that disappears and returns. https://svnweb.freebsd.org/changeset/base/325011

Previously, this didn't work because L2ARC devices' labels don't contain pool GUIDs. Modify zfsd so that the pool GUID won't be required.

Fix zpool_read_all_labels when vfs.aio.enable_unsafe=0. https://svnweb.freebsd.org/changeset/base/324991

Previously, zpool_read_all_labels was trying to do 256KB reads, which are greater than the default MAXPHYS, and, therefore, must go through the slow, unsafe AIO path. Shrink these reads to 112KB so they can use the safe, fast AIO path instead.

Fix the error message when creating a zpool on a too-small device. https://svnweb.freebsd.org/changeset/base/324940

Don't check for SPA_MINDEVSIZE in vdev_geom_attach when opening by path. It's redundant with the check in vdev_open, and failing to attach here results in the wrong error message being printed.

Add vfs_zfs.abd_chunk_size tunable. https://svnweb.freebsd.org/changeset/base/323797

It is reported that the default value of 4KB results in a substantial memory use overhead (at least, on some configurations). Using 1KB seems to reduce the overhead significantly.

Add sysctls for arc shrinking and growing values. https://svnweb.freebsd.org/changeset/base/323051

The default value for arc_no_grow_shift may not be optimal when using several GiB ARC. Expose it via sysctl allows users to tune it easily. Also expose arc_grow_retry via sysctl for the same reason. The default value of 60s might, in case of intensive load, be too long.

msdosfs(5): Reflect READONLY attribute in file mode. https://svnweb.freebsd.org/changeset/base/326031

Msdosfs allows setting READONLY by clearing the owner write bit of the file mode. In msdosfs_getattr, intuitively reflect that READONLY attributes to userspace in the file mode.

Use taskqueue(9) to do writes/commits to mirrored DSs concurrently. https://svnweb.freebsd.org/changeset/base/324676

When the NFSv4.1 pNFS client is using a Flexible File Layout specifying mirrored Data Servers, it must do the writes and commits to all mirrors. This change modifies the client to use a taskqueue to perform these writes and commits concurrently. The number of threads can't be changed for taskqueue(9), so it is set to 4 * mp_ncpus by default, but this can be overridden by setting the sysctl vfs.nfs.pnfsiothreads.

STEVEN KREUZER is a FreeBSD Developer and Unix Systems Administrator with an interest in retro-computing and air-cooled Volkswagens. He lives in Queens, New York, with his wife, daughter, and dog.

2018 Events Calendar

The following BSD-related conferences will take place during March and April 2018.

AsiaBSDCon • March 8–11 • Tokyo, Japan

https://2018.asiabsdcon.org/ • This is the annual BSD technical conference for users and developers on BSD-based systems. It provides several days of workshops, presentations, and a Developer Summit. Registration is required for this event.

SCALE • March 8–11 • Pasadena, CA

http://www.socallinuxexpo.org/scale/16x • The 16th annual Southern California Linux Expo will once again provide several FreeBSD-related presentations and a FreeBSD booth in the expo area. This event requires registration at a nominal fee.

FOSSASIA Summit 2018 • March 22–25 • Singapore

https://2018.fossasia.org/ • The FOSSASIA Summit, Asia's leading Open Technology conference for developers, startups, and IT professionals, will take place from March 22-25, 2018, at the Lifelong Learning Institute Singapore.

ZFS User Conference • April 19 & 20 • Norwalk, CT

http://zfs.datto.com/ • The second annual conference focuses on the deployment, administration, and tuning of the ZFS filesystem. The conference consists of two days of talks and networking. This event requires registration at a nominal fee.